Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Machine Learning-Based Waiting Time Prediction for Priority Queues with Preemptive and Non-Preemptive Service Rules

Sahil Singh Bhandari
UM-DAE Center For Excellence in basic sciences
T.G. Deepak
deepak@iist.ac.in

Indian Institute of Space Science and Technology

Research Article

Keywords: Priority queuing systems, Waiting time prediction, Machine learning, Time-varying systems

Posted Date: June 29th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10031674/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Highlights

- Machine-learning models are used to predict transient waiting times in multi-class priority queuing systems.
- Recurrent neural networks achieve the highest accuracy, especially in time-varying and preemptive settings.
- Tree-based models provide stable and reliable performance across different service disciplines.
- Forcing priority-based splitting in tree models provides only marginal improvement.
- Significant waiting-time prediction is possible using only queue length at arrival and customer priority.
- The approach supports practical real-time prediction in priority-based service systems with limited data.

Machine Learning-Based Waiting Time Prediction for Priority Queues with Preemptive and Non-Preemptive Service Rules

Sahil Singh Bhandari[1] and T.G. Deepak[2*†]

[1]School of Mathematical Sciences, UM-DAE Center For Excellence In Basic Sciences,Mumbai,400098, India.
[2]Department of Mathematics, Indian Institute of Space Science and Technology, Thiruvananthapuram, 695547,India.

*Corresponding author(s). E-mail(s): deepak@iist.ac.in;
Contributing authors: sahil.singhbhandari@cbs.ac.in;
†These authors contributed equally to this work.

Abstract

This paper studies how machine learning can predict waiting times in time-varying, multi-class priority queuing systems. Five models are evaluated: Gradient Boosting Machine (GBM), Random Forest (RF), their priority-aware versions, and a Recurrent Neural Network (RNN). These models are tested under three service settings: non-preemptive service, preemptive service with repeated service times, and preemptive service with resumed service times. A 200-day synthetic dataset is used, generated from nonstationary arrivals and class-dependent phase-type service-time distributions. Performance is evaluated using MAE, MSE, RMSE, and $\boldsymbol{R^2}$. Results show that the RNN gives the highest accuracy for all the settings. Tree-based models also perform well and remain stable, while priority-based splitting provides only small improvements. Importantly, accurate waiting-time prediction can be achieved using only queue length at arrival and customer priority. This highlights the practical usefulness of the approach for real-world priority-based service systems.

Keywords: Priority queuing systems, Waiting time prediction, Machine learning, Time-varying systems

1 Introduction

Priority queues are widely used in service systems where customers or tasks are ranked by importance or urgency. Unlike the usual first-come-first-served (FCFS) rule, higher-priority customers are served earlier. This is important in settings where delays have different consequences.

Such systems appear in many real-world applications. In healthcare, emergency cases are treated before regular patients. In call centers, urgent or VIP requests are prioritized. In manufacturing and logistics, critical tasks interrupt routine operations. Similar priority rules are used in computing systems, communication networks, and transportation.

Predicting waiting times in these systems is very important for planning and decision-making. However, it is difficult in practice. Classical queuing models provide exact results only for simple and stationary systems. Real systems are more complex: arrival rates change over time, service times vary, and preemption allows interruptions. In such cases, a customer's waiting time-especially for lower-priority classes-depends not only on the current queue but also on future high-priority arrivals. This makes analytical prediction hard.

Machine learning (ML) provides a practical alternative. ML models can learn complex patterns directly from data without requiring detailed assumptions about the system. This is especially useful when only limited information is available.

In many real systems, only basic data is recorded, such as the queue length at arrival and the customer's priority. Detailed information like arrival rate functions or service-time distributions is often unknown. Therefore, an important question is whether accurate waiting-time prediction is possible using only such limited inputs.

In this work, we study three common service disciplines:

1. **Non-preemptive service**: service continues without interruption.
2. **Preemptive service (restart)**: interrupted jobs restart from the beginning.

3. **Preemptive service (resume)**: interrupted jobs continue from where they stopped.

We evaluate five models: Gradient Boosting Machine (GBM), Random Forest (RF), their priority-split variants, and a Recurrent Neural Network (RNN) with LSTM units. These variants are included to evaluate whether explicit priority-aware structure improves prediction. A 200-day synthetic dataset is generated using nonstationary arrivals and phase-type service-time distributions, allowing realistic and flexible system behavior under all service disciplines. Model performance is evaluated under two settings: (1) using all available features, and (2) using only two simple features-**queue length at arrival** and **priority**. The results show that even with this limited information, ML models-especially the RNN-can achieve strong prediction accuracy. This highlights the potential of simple, data-driven methods for real-world priority-based service systems.

2 Related Work

Priority queues have been studied extensively in queuing theory because they arise naturally in systems where some customers or tasks must receive preferential treatment. Classical references such as Gross et al. (2008) provide a broad introduction to waiting-time analysis, service disciplines, and multi-class queuing systems, while Takagi (1991) presents detailed treatments of non-preemptive, preemptive-repeat, and preemptive-resume priority queues. These studies establish the theoretical foundations for understanding how priority rules influence waiting times and overall system performance.

Priority-based service mechanisms are also important in communication and computer systems. For example, the generalized processor sharing (GPS) framework developed by Parekh and Gallager (1993) demonstrates how different customer classes can receive differentiated service while maintaining performance guarantees. Similar

priority mechanisms appear in healthcare systems, transportation networks, manufacturing systems, emergency services, and call centers, where some customers or tasks must be processed ahead of others due to urgency or operational requirements.

Another important area of research concerns time-varying queues. In many real systems, arrival rates fluctuate throughout the day, making steady-state assumptions unrealistic. The pointwise stationary approximation introduced by Whitt (1991) remains one of the most widely used approaches for analyzing such nonstationary systems. Applications in healthcare and service operations further highlight the practical importance of waiting-time estimation under dynamic conditions. For example, Cayirli and Veral (2003) review appointment scheduling models in outpatient clinics, while Brown et al. (2005), Gans et al. (2003), and Aksin et al. (2007) discuss the challenges of managing large-scale service systems with time-varying demand, customer waiting, and service-level requirements.

Predicting waiting times has attracted significant attention because accurate delay estimates can improve customer satisfaction, resource allocation, and operational decision making. In many service environments, waiting-time information influences customer behavior, abandonment decisions, and perceptions of service quality. A comprehensive review of delay-information sharing in service systems is provided by Ibrahim (2018), which highlights the importance of accurate waiting-time estimation for both customers and system managers. Recent studies have further demonstrated that forecasting future queue states can support staffing and resource-allocation decisions. For example, Bhandari and Deepak (2026) used simulation-based machine learning models to predict transient queue congestion and derive staffing recommendations in a healthcare service environment.

Traditional queuing models provide valuable analytical insight into waiting-time behavior, but they often require detailed assumptions regarding arrival processes, service distributions, and system structure. In complex real-world environments, these

assumptions may be difficult to verify or may not hold in practice. As a result, data-driven approaches have become increasingly popular as a complementary tool for predicting delays directly from observed system data.

Machine learning methods have increasingly been applied to waiting-time prediction in service systems. In healthcare settings, machine-learning models have been used to predict emergency department and outpatient waiting times using operational and patient-flow variables Sun et al. (2020); McGuire et al. (2022). More generally, data-driven learning approaches have also been proposed for predicting queuing performance measures directly from observed system data, providing an alternative to traditional analytical approximations Baron et al. (2024). These studies demonstrate that supervised learning methods can successfully learn nonlinear relationships between operational variables and observed delays. However, most existing work focuses on single-class systems, steady-state environments, or settings where future arrivals have little influence on the waiting time of a customer already in the system. Consequently, the unique challenges associated with priority queues have received comparatively little attention. This gap is particularly important because customer waiting and delay information can significantly influence system performance and customer behavior Ibrahim et al. (2017). Among machine learning approaches, tree-based ensemble methods and recurrent neural networks are particularly relevant. Gradient Boosting Machines (GBM) Friedman (2001) and Random Forests (RF) Breiman (2001) are widely used because they can capture nonlinear relationships, model complex feature interactions, and remain robust in the presence of noisy data. Their strong predictive performance and relative interpretability have made them popular in many operational forecasting applications.

Recurrent neural networks, especially Long Short-Term Memory (LSTM) networks Hochreiter and Schmidhuber (1997); Goodfellow et al. (2016), provide a natural

framework for modeling temporal dependence. Because waiting times in nonstationary queues depend on evolving congestion levels and arrival patterns, LSTMs are well suited for learning such sequential relationships. Previous studies have reported strong performance of LSTM-based models for forecasting and delay prediction when historical system trajectories contain useful information. However, these models often require larger training datasets and may provide less interpretability than tree-based approaches.

Despite these advances, most machine learning studies treat customer priority simply as another input feature. The learning architecture itself is rarely modified to reflect the structure of priority-based service rules. This raises an interesting question: can explicitly incorporating queuing-domain knowledge into the learning process improve predictive performance? To investigate this question, the present study considers priority-split variants of GBM and RF, where customers are first separated according to priority class before model training. Rather than proposing these variants as universally superior models, they are included to evaluate whether explicit priority-aware structure provides any measurable benefit over standard machine-learning approaches.

To the best of our knowledge, there has been little work comparing standard machine-learning models with priority-aware variants for predicting transient customer-level waiting times in multi-class priority queues. Existing queuing studies primarily focus on analytical performance measures, while existing machine-learning studies generally do not address transient waiting-time prediction under preemptive and non-preemptive priority disciplines where future arrivals can influence current waiting times.

The present work addresses this gap by combining simulation-based data generation with machine-learning prediction. Specifically, we develop a unified framework

for generating datasets from non-preemptive, preemptive-repeat, and preemptive-resume priority queues with time-varying arrivals and phase-type service distributions. We then compare several machine-learning models, including GBM, RF, priority-split variants, and LSTM-based RNNs, for predicting transient customer-level waiting times. This positions the study at the intersection of classical queuing theory and modern data-driven prediction for priority-based service systems.

3 Data Generation

3.1 Simulation of 200-Day Priority Queue Dataset

To construct the synthetic dataset used in this study, we simulated 200 independent days of a two-class priority queue operating under a time-varying $G(t)/G/1$ environment. The data generation design is inspired by the sampling methodology adopted in *Supervised Machine Learning for Solving the GI/GI/1 Queue* (see: Baron et al. (2024)), where nonstationary arrivals are generated through controlled piecewise-homogeneous intervals. Following this approach ensures that the simulated dataset captures realistic transient behavior and sufficiently rich variation for training data-driven predictive models.

The simulation procedure used in this work is summarized below.

1. **Time interval discretization:** The 6-hour operating window $[0, 360]$ minutes was divided into 100 randomly sized sub-intervals. This piecewise segmentation follows the idea of approximating a time-varying arrival process by stitching together locally stationary regions, similar to the approach used in GI/GI/1 sampling (Baron et al. (2024)).
2. **Allocation of customers across intervals:** Each simulated day contains 180 customers. These customer positions were assigned proportionally across the 100 sub-intervals, producing a flexible and non-uniform arrival pattern. Two priority

classes were defined:

$$150 \text{ Priority-2 customers}, \qquad 30 \text{ Priority-1 customers}.$$

3. **Generation of time-varying arrival rates:** For each interval, mean arrival rates were sampled independently for both priority classes:

$$\lambda_2(t) \sim \mathcal{U}(0.3, 0.5), \qquad \lambda_1(t) \sim \mathcal{U}(0.06, 0.09).$$

This produces a nonstationary arrival stream that varies across the day, consistent with the sampling strategy used to model time-varying GI/GI/1 systems (Baron et al. (2024)).

4. **Inter-arrival time generation via PH distributions:** For each sub-interval, inter-arrival times were sampled from a Phase-Type (PH) distribution following the supervised sampling framework of Baron et al. (2024), PH parameters were generated to span a broad family of non-negative distributions, whose mean matches the interval-specific arrival rate:

$$\mathbb{E}[T_{ia}(t)] = \frac{1}{\lambda(t)}.$$

The use of PH distributions provides flexibility in shape, enabling the modeling of over dispersion and variability beyond exponential arrivals-an approach consistent with the GI/GI/1 sampling framework (Baron et al. (2024)).

5. **Service time generation using PH distributions:** Service times were likewise generated from PH distributions, tuned to match class-dependent target means:

- **Priority 1 (high priority):**

$$\text{lower} = 1,\ \text{upper} = 1.5,\ \text{mean} = 1.3.$$

- **Priority 2 (regular):**

$$\text{lower} = 1.5,\ \text{upper} = 2,\ \text{mean} = 1.8.$$

Using PH distributions for service modeling allows capturing realistic variability and non-exponential behaviors characteristic of GI-type service processes.

The overall design ensures that the synthetic dataset exhibits sufficiently rich dynamics-including nonstationarity, class heterogeneity, and variability in service durations-making it suitable for training and evaluating data-driven prediction models for priority queues.

3.2 Simulation of Waiting and Departure Times

To generate the transient waiting times and departure times for each customer, we simulate the dynamics of a single-server two-class priority queue for each of the 200 days of data. The customers belong to either Priority 1 (high priority) or Priority 2 (regular priority). The simulations are performed separately under three service disciplines: (i) non-preemptive service, (ii) preemptive service with repeated service times, and (iii) preemptive service with resumed service times. In all settings, Priority 1 customers always receive higher precedence than Priority 2 customers. The simulation is executed independently for each day after sorting all arrivals in increasing order of time.

3.2.1 Non-Preemptive Priority Service

In the non-preemptive model, once the server begins serving a customer, the service continues without interruption, even if a higher-priority customer arrives. Two FCFS queues are maintained-one for Priority 1 and one for Priority 2 customers. At any service completion instant, a Priority 1 customer is selected if available; otherwise a Priority 2 customer is selected.

For each arriving customer with arrival time A_i and service duration S_i:

1. If the server is idle at A_i, the customer begins service immediately.
2. If the server is busy, the customer joins the appropriate priority queue.
3. When the server becomes free, the next customer is selected according to:

$$\text{Priority 1 (FCFS)} \succ \text{Priority 2 (FCFS)}.$$

4. The waiting time is computed as $W_i = \max(0, \text{start}_i - A_i)$, and the departure time is $\text{Dep}_i = \text{start}_i + S_i$.

3.2.2 Preemptive Priority with Repeated Service Times

In the preemptive-restart discipline, a Priority 1 arrival immediately interrupts a Priority 2 customer in service. The interrupted Priority 2 customer returns to the queue and must *restart* its entire service upon resumption; any service already completed is discarded. Priority 1 customers never preempt each other.

The event logic is:

1. If a Priority 1 arrival occurs while serving a Priority 2 customer, the ongoing service is stopped and discarded.
2. The preempted Priority 2 customer is placed back at the front of the Priority 2 queue with its *full original service time*.
3. The arriving Priority 1 customer immediately begins service at its arrival time.

4. If the server becomes idle, the next customer is selected with Priority 1 precedence over Priority 2, restarting service if the customer was previously preempted.

Waiting and departure times are recorded after each service completion.

3.2.3 Preemptive Priority with Resumed Service Times

In the preemptive-resume discipline, a Priority 1 arrival interrupts a Priority 2 customer, but the partially completed service is *not lost*. Instead, the remaining service time is computed and the customer resumes exactly where it left off once it re-enters service.

For a preempted Priority 2 customer:

$$S_i^{\text{rem}} = S_i^{\text{orig}} - (A_{\text{P1}} - \text{start}_i)\,,$$

where A_{P1} is the arrival time of the interrupting Priority 1 customer.

The event logic is:

1. A Priority 1 arrival preempts an ongoing Priority 2 service.
2. The remaining service time of the Priority 2 customer is computed and the customer is placed back into the Priority 2 queue.
3. The Priority 1 customer begins service immediately.
4. After service completion, Priority 1 customers are served first; if none remain, the server resumes the next Priority 2 customer from its remaining service time.

In all three models, the simulation proceeds by repeatedly identifying the next event (either an arrival or a service completion) and updating the system state until all customers for a given day are served. The resulting waiting and departure times form the dataset used for the machine learning prediction models.

3.3 Computation of Queue Length at Arrival

For each customer, the queue length observed at the instant of arrival is computed using the simulated arrival and departure times. After generating the waiting and departure times, we convert the timestamp variables into numerical time durations to enable direct comparison. For each day, the customers are sorted chronologically by arrival time, and the queue length for customer i is calculated as the number of customers who arrived before customer i and whose service had not yet completed at customer i's arrival instant.

Let A_i and D_j denote the arrival time of customer i and the departure time of customer j, respectively. Then, for each customer i in a given day, the queue length at arrival is computed as

$$Q_i = \sum_{j<i} \mathbf{1}\{D_j > A_i\},$$

where the indicator $\mathbf{1}\{D_j > A_i\}$ equals 1 if customer j is still in service when customer i arrives, and 0 otherwise. The resulting sequence $\{Q_i\}$ provides the instantaneous queue lengths used as an input feature for the machine learning models.

4 Models

4.1 Feature Configurations Used for Model Training

To evaluate the robustness and practical feasibility of the predictive models, we consider two different feature sets during training:

1. **Four-feature setting:** arrival time, inter-arrival time, priority class, and queue length at arrival.
2. **Two-feature setting:** priority class and queue length only.

The first configuration represents a richer information environment, while the second reflects a minimal and practically attractive setting where only operationally

observable queue-state information is required. The same train-test partitioning was used across all experiments, while model-specific hyperparameters were selected using the procedures described in the appendix (for 4 feature model). This allows a direct comparison of predictive performance across different levels of input information.

4.2 Gradient Boosting Machine (GBM)

Gradient Boosting Machine (GBM) is an ensemble learning method that constructs a prediction model in the form of an additive combination of weak decision-tree learners. GBM iteratively fits regression trees to the residuals of previous iterations, thereby improving the model in directions that reduce the loss function. Due to its ability to model non-linear relationships, capture complex interactions, and provide feature importance scores, GBM represents a strong baseline for waiting-time prediction in queuing systems.

4.2.1 Feature Construction

For each customer, the following explanatory variables are used as model inputs:

- **Priority** (1 or 2), indicating customer class.
- **Arrival minutes**, obtained by converting the timestamp of arrival into the minute of the day using

$$\text{Arrival_Minutes} = 60 \times \text{hour} + \text{minute}.$$

- **Queue length at arrival**, computed as the number of customers still in service at the instant of arrival (explained in section 3.3).
- **inter-arrival time**, derived separately for Priority 1 and Priority 2 customers from PH-distributed inter-arrival samples, and merged into the main dataset using day-wise and class-wise indexing.

Let X_i denote the feature vector for customer i, and let Y_i denote the corresponding waiting time. The dataset is assembled as

$$X_i = \big(\text{Priority}_i, \text{ArrivalMinutes}_i, \text{QueueLength}_i, \text{inter-arrival}_i\big), \qquad Y_i = \text{WaitingTime}_i.$$

4.2.2 Training Procedure

The complete dataset is sorted chronologically and divided into an 80% training set and a 20% test set using a temporal cutoff to avoid information leakage. Hyperparameters were selected based on the sensitivity and bias–variance analyses reported in Appendices A–C. The final `GBM` configurations used the estimator counts and tree depths identified as providing the best generalization performance. The model learns a sequence of regression trees $\{h_m(X)\}$ combined as

$$\hat{Y}(X) = \sum_{m=1}^{M} \nu \, h_m(X),$$

where M is the number of boosting stages and ν is the learning rate.

After prediction, all negative values (which are not meaningful in queuing contexts) are truncated to zero:

$$\hat{Y}_i = \max(0, \hat{Y}_i).$$

4.2.3 Performance Evaluation

Model performance is assessed on the test dataset using four metrics:

$$\begin{aligned}
\text{MAE} &= \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|, \\
\text{MSE} &= \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2, \\
\text{RMSE} &= \sqrt{\text{MSE}}, \\
R^2 &= 1 - \frac{\sum_i (Y_i - \hat{Y}_i)^2}{\sum_i (Y_i - \bar{Y})^2},
\end{aligned}$$

These metrics provide complementary views of absolute error, squared error, relative error, and overall explanatory power.

4.3 Forced Priority-Split GBM

In priority queuing systems, customers of different classes often experience fundamentally different waiting-time dynamics. For example, high-priority customers (P1) may receive preferential access to service capacity, whereas low-priority customers (P2) may experience preemption, delays, or longer service times. To investigate whether explicit priority-based separation improves waiting-time prediction, we also consider a `Forced Priority-Split Gradient Boosting Machine` (FS-GBM), in which the dataset is partitioned according to customer priority before model training. Unlike the standard GBM that learns a single mapping $X \mapsto Y$, the FS-GBM trains two separate GBM models-one for each priority class-after routing customers through a simple decision rule.

4.3.1 Priority-Based Router

A decision stump (depth-1 decision tree) is used as a deterministic router:

$$\text{Router}(X_i) = \begin{cases} 1, & \text{if Priority}_i = 1, \\ 2, & \text{if Priority}_i = 2, \end{cases}$$

which effectively hard-splits the dataset by customer class. This is equivalent to assuming that the relationship between features and waiting time is priority-specific and cannot be adequately captured by a single shared model.

4.3.2 Priority-Wise GBM Training

Two independent GBM models are trained, one for each priority level:

$$\hat{Y}^{(p)}(X) = \sum_{m=1}^{M_p} \nu_p \, h_m^{(p)}(X), \qquad p \in \{1, 2\},$$

where M_p, ν_p, and the regression-tree depths are tuned separately for each class. Empirically, we found that P1 customers required a deeper ensemble (800 estimators, learning rate 0.01), while P2 customers were adequately modeled using a smaller but deeper ensemble (300 estimators, learning rate 0.05). This asymmetric configuration reflects differences in the variability of waiting times across priority classes.

For training, the dataset is split chronologically by day: the first 80% of days are used for training and the last 20% for testing. The same input features used in the standard GBM are retained:

$$X_i = (\text{Priority}_i, \text{ArrivalMinutes}_i, \text{QueueLength}_i, \text{inter-arrival}_i).$$

Priority is removed from the feature set inside each submodel, since customers are already grouped by class.

4.3.3 Hybrid Prediction Function

At inference time, the router sends each test observation to its priority-specific model:

$$\hat{Y}_i = \begin{cases} \hat{Y}^{(1)}(X_i), & \text{if Priority}_i = 1, \\ \hat{Y}^{(2)}(X_i), & \text{if Priority}_i = 2, \end{cases}$$

and negative predictions are truncated to zero to maintain physical validity in queuing contexts:

$$\hat{Y}_i \leftarrow \max(0, \hat{Y}_i).$$

4.3.4 Motivation and Observed Performance

The motivation for FS-GBM stems from the expectation that allowing the model to learn two *separate functional relationships*-one for each priority class-would yield improved predictive accuracy. Priority-based hard separation is commonly beneficial in systems where the service discipline creates distinct statistical behaviors across customer types.

However, our empirical results show that FS-GBM does not significantly improve over the standard GBM in this application. Despite the structural differences introduced by priority routing, the gain in predictive accuracy was marginal for both priority classes. This suggests that the standard GBM is already capable of implicitly learning priority-specific nonlinearities from the full feature set, and that explicit hard separation does not offer additional advantage for the queuing dynamics in our dataset.

The FS-GBM results are nevertheless included for methodological completeness and to illustrate the trade-off between model complexity and predictive benefit in priority queue prediction tasks.

4.4 Recurrent Neural Network (RNN) Model

Recurrent Neural Networks (RNNs), particularly Long Short-Term Memory (LSTM) architectures, are well suited for sequence modeling tasks in which temporal dependencies play a central role. In priority queuing systems, the waiting time experienced by each customer is not only a function of their own arrival attributes but also of the congestion state shaped by the sequence of customers arriving before them. This motivates the use of an LSTM-based RNN to learn temporal patterns within each service day.

4.4.1 Feature Representation and Scaling

For each customer, the input feature vector includes the following variables:

$$X_i = (\text{Priority}_i,\ \text{QueueLength}_i,\ \text{inter-arrivalTime}_i)\,.$$

These features capture (1) the service discipline, (2) real-time congestion, and (3) stochastic arrival variability. All features are scaled using a Min-Max normalization to the range $[0, 1]$. The target variable, waiting time, is scaled using the same transformation. Scaling is essential for stabilizing gradient updates during LSTM training.

4.4.2 Sequence Construction

Each simulation day consists of 180 customers. The data are therefore reshaped into daily sequences of length 180:

$$\mathbf{X}^{(d)} \in \mathbb{R}^{180\times 3}, \qquad \mathbf{Y}^{(d)} \in \mathbb{R}^{180\times 1},$$

where each sequence represents the chronological order of arrivals. Days with fewer than 180 observations are excluded to maintain uniform input dimensions.

The full dataset is converted into

$$N_{\text{days}}$$

such sequences, which form the input to the RNN.

4.4.3 Train-Test Splitting

To preserve temporal consistency, we adopt a chronological split: the first 80% of days are used for training and the remaining 20% for testing. This ensures that predictions are evaluated on future data, aligning with realistic deployment scenarios.

4.4.4 Network Architecture

The RNN model is a stacked LSTM architecture implemented in TensorFlow/Keras. The model consists of:

- The baseline architecture consisted of a stacked LSTM network. The number of LSTM units was selected separately for each service discipline based on the sensitivity analyses reported in the appendix. The final architectures used 64 units for the non-preemptive and preemptive-repeat systems and 128 units for the preemptive-resume system,

- a dropout layer ($p = 0.2$) to reduce overfitting,
- a second LSTM layer with 32 units,
- another dropout layer ($p = 0.2$),
- a TimeDistributed dense layer with 16 ReLU units,
- a TimeDistributed dense output layer with a Softplus activation to guarantee non-negative waiting-time predictions.

Formally, the model maps each daily sequence to a predicted waiting-time sequence:

$$\widehat{\mathbf{Y}}^{(d)} = f_\theta\left(\mathbf{X}^{(d)}\right),$$

where f_θ denotes the learned LSTM representation.

The network is trained for 500 epochs using the Adam optimizer and mean squared error (MSE) loss. A validation split of 20% is used to monitor convergence.

4.4.5 Prediction and Postprocessing

Predictions generated by the RNN are initially in scaled form. They are transformed back to the original scale using the inverse Min-Max transform:

$$\hat{y}_i = \text{InverseScale}(\hat{y}_{i,\text{scaled}}).$$

Finally, predictions are truncated to ensure non-negativity:

$$\hat{y}_i \leftarrow \max(0, \hat{y}_i).$$

Evaluation metrics are computed over the flattened test sequences: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), coefficient of determination (R^2), and Mean Absolute Percentage Error (MAPE).

4.5 Random Forest Models

Random Forests (RF) are powerful ensemble learning methods that construct a collection of decision trees and aggregate their predictions through averaging. RF models are particularly effective in nonlinear regression tasks with interacting features and heterogeneous data patterns. In the context of priority queues, waiting times depend jointly on customer attributes, temporal arrival characteristics, and evolving system congestion. The RF framework is therefore a natural candidate for modeling such complex relationships.

4.5.1 Feature Engineering

The feature vector for each customer is defined as

$$X_i = (\text{Priority}_i,\ \text{ArrivalMinutes}_i,\ \text{QueueLength}_i,\ \text{inter-arrivalTime}_i)\,,$$

where:

- **Priority** encodes the service discipline,
- **ArrivalMinutes** captures the time-of-day effect,
- **QueueLength** represents instantaneous congestion,
- **inter-arrival time** models short-term temporal dynamics.

The target variable is the customer-level waiting time. Days are split chronologically, with the first 80% used for training and the last 20% reserved for testing.

4.5.2 Standard Random Forest Model

Hyperparameters were selected using the sensitivity analyses reported in the appendix. The final RF models used moderate tree depths (approximately depth 8) and

sufficiently large ensembles to ensure stable performance. The model learns a mapping

$$\hat{Y} = f_{\mathrm{RF}}(X),$$

where each tree captures different local patterns in the data, and the ensemble averages across them to reduce variance. Because waiting times are inherently non-negative, predictions are post-processed as

$$\hat{Y}_i \leftarrow \max(0, \hat{Y}_i).$$

Performance is evaluated through MAE, MSE, RMSE and R^2. The RF model serves as a competitive baseline due to its ability to capture nonlinear and interaction effects without requiring feature scaling or time-sequence structure.

4.5.3 Forced Priority-Split Random Forest (FS-RF)

Although a single RF can implicitly learn differences between priority classes, we also evaluate a *forced-split* version included to test the hypothesis that each priority class exhibits structurally different queuing behavior.

Priority Router.

A decision stump,

$$\mathrm{Router}(X_i) = \begin{cases} 1, & \mathrm{Priority}_i = 1, \\ 2, & \mathrm{Priority}_i = 2, \end{cases}$$

is used to deterministically separate the dataset. This routing mechanism mirrors the operational reality that P1 and P2 customers follow different waiting-time dynamics.

Priority-Wise RF Training.

Two RF models are trained independently:

$$\hat{Y}^{(p)} = f_{\mathrm{RF}}^{(p)}(X), \qquad p \in \{1, 2\},$$

each with its own hyperparameters:

$$\begin{aligned} &\text{P1:} \quad 150 \text{ trees, max depth } 6, \\ &\text{P2:} \quad 120 \text{ trees, max depth } 7. \end{aligned}$$

Within each class-specific model, the *Priority* feature is removed from the input because customers are already split by class.

Hybrid Prediction.

For a test observation,

$$\hat{Y}_i = \begin{cases} f_{\mathrm{RF}}^{(1)}(X_i), & \text{if Priority}_i = 1, \\ f_{\mathrm{RF}}^{(2)}(X_i), & \text{if Priority}_i = 2. \end{cases}$$

Negative values are truncated:

$$\hat{Y}_i \leftarrow \max(0, \hat{Y}_i).$$

4.5.4 Discussion

The forced-split RF model was motivated by the expectation that modeling priority classes separately would reduce cross-class noise and enhance prediction accuracy. Surprisingly, empirical results show that FS-RF provides only marginal improvements

over the standard RF. This indicates that the single RF model is already capable of learning strong nonlinear interactions involving priority without explicit separation. Nevertheless, the forced-split variant provides valuable insight into the structural behavior of tree-based models in priority queue prediction tasks, and helps quantify the degree to which priority-specific patterns contribute to predictive performance.

5 Results

This section reports the comparative performance of the five predictive models- GBM, FS-GBM, RF, FS-RF, and RNN-under three queuing mechanisms:(i) non-preemptive service, (ii) preemptive service with restarted processing, and (iii) preemptive service with resumed processing. For each mechanism, we evaluate models under two cases:

1. **Four-feature setting:** arrival time, interarrival time, queue length at arrival, and priority.
2. **Two-feature setting:** queue length at arrival and priority only.

Prediction performance is assessed using MAE, MSE, RMSE, and R^2. Tables 1-6 summarize results for all six combinations.

Table 1: Performance under non-preemptive service using four features.

Model	R^2	MAE	MSE	RMSE
GBM	0.942	1.003	3.210	1.792
FS-GBM	0.942	1.0018	3.228	1.7966
RNN	0.987	0.643	0.697	0.835
RF	0.941	1.005	3.287	1.813
FS-RF	0.942	0.993	3.209	1.791

Table 2: Performance under preemptive-repeat service using four features.

Model	R^2	MAE	MSE	RMSE
GBM	0.937	1.748	7.870	2.805
FS-GBM	0.9366	1.7205	7.880	2.8065
RNN	0.988	0.908	1.508	1.228
RF	0.935	1.739	8.088	2.844
FS-RF	0.936	1.717	7.908	2.812

Table 3: Performance under preemptive-resume service using four features.

Model	R^2	MAE	MSE	RMSE
GBM	0.927	1.207	4.277	2.068
FS-GBM	0.9267	1.1913	4.290	2.0717
RNN	0.987	0.496	0.526	0.725
RF	0.927	1.201	4.397	2.097
FS-RF	0.927	1.184	4.285	2.070

Table 4: Performance under non-preemptive service using two features.

Model	MAE	MSE	RMSE	R^2
GBM	1.0160	3.2270	1.7960	0.9420
FS-GBM	1.0082	3.2016	1.7893	0.9424
RNN	0.8360	1.4510	1.2040	0.9740
RF	1.0090	3.2184	1.7940	0.9420
FS-RF	1.0100	3.2250	1.7960	0.9420

5.1 Results Using Four Features

Across all service disciplines, the RNN achieves the highest predictive performance. In the non-preemptive case, the RNN obtains the largest R^2 values (above 0.98) and the lowest MAE and RMSE, as shown in Table 1. GBM and RF models also perform competitively, achieving R^2 values in the 0.93-0.94 range.

Under preemption with repeated service (Table 2), the RNN continues to outperform all tree-based models with a wide margin, reflecting its ability to model the sequential dynamics introduced by service interruptions and restarts.

Table 5: Performance under preemptive-repeat service using two features.

Model	MAE	MSE	RMSE	R^2
GBM	1.7590	7.9880	2.8260	0.9360
FS-GBM	1.7298	7.9614	2.8216	0.9360
RNN	0.9080	1.5080	1.2280	0.9880
RF	1.7300	7.9637	2.8220	0.9360
FS-RF	1.7300	7.9630	2.8220	0.9360

Table 6: Performance under preemptive-resume service using two features.

Model	MAE	MSE	RMSE	R^2
GBM	1.2450	4.3330	2.0820	0.9260
FS-GBM	1.1913	4.2832	2.0696	0.9269
RNN	0.8490	1.6200	1.2730	0.9720
RF	1.1910	4.2725	2.0670	0.9270
FS-RF	1.1910	4.2770	2.0680	0.9270

The preemptive-resume setting (Table 3) presents the highest difficulty because waiting times depend on the remaining service time of interrupted tasks. Even in this most complex scenario, the RNN provides the best performance, while GBM and RF show a noticeable decrease in accuracy.

Interestingly, the forced priority-split variants (FS-GBM and FS-RF) show only minor and inconsistent improvements over the base GBM/RF models. This indicates that priority is already sufficiently represented as a feature, and hard-splitting is not necessary for tree-based architectures.

5.2 Results Using Two Features

A central objective of this study is to understand whether feasible waiting-time predictions are possible using minimal information. When the feature set is reduced to only *queue length at arrival* and *priority*, all models retain most of their predictive capability.

Table 4 shows that even with only two features, the RNN achieves strong performance for the non-preemptive system, with MAE and RMSE values only slightly higher than the four-feature case. GBM and RF models also remain stable, demonstrating that a large portion of the predictive structure is encoded directly through queue length and priority.

Under preemptive-repeat and preemptive-resume service (Tables 5 and 6), the expected performance drop appears, but the RNN continues to outperform all other models. The decrease in accuracy across models remains moderate-confirming that the temporal features (arrival time and interarrival time) improve accuracy but are not strictly required for reasonably good predictions.

Overall, these results should be interpreted positively: **Effective transient waiting-time prediction in priority queues is feasible even with minimal features**, making ML-based queuing prediction practical for real-world deployment.

5.3 Cross-Model and Cross-Setting Comparison

Three consistent patterns emerge:

1. **RNN is the most effective predictor** across all service disciplines and feature settings.
2. **Tree-based models (GBM, RF) perform reliably and competitively**, but their ability to learn temporal structure is limited in preemptive settings.
3. **Forced priority splitting does not provide substantial gains**, confirming that priority information is already well captured by the features.

5.4 Graphical Interpretation

In addition to the numerical metrics, Figures1 and 2 provide a visual comparison of predicted versus actual waiting times for one representative day under all models and all queuing disciplines. Across both feature configurations (4-feature and 2-feature),

the scatter plots exhibit a clear upward trend closely aligned with the $y = x$ reference line, indicating that all models capture the fundamental waiting-time structure.

Among the models, the RNN produces the tightest clustering around the diagonal, reflecting its superior ability to represent temporal patterns and queue evolution. GBM and RF also show strong linear alignment, though with slightly greater dispersion, especially under the preemptive scenarios. The priority-split variants yield plots that are visually similar to their baseline counterparts, further confirming that explicit priority-based splitting does not meaningfully alter predictive behavior.

A notable observation is that-even with only two input features (Queue Length and Priority)-the predicted-actual points remain well structured, and the overall linear pattern is preserved across all queue types. This visually reinforces the numerical results showing that feasible waiting-time prediction is feasible even under minimal information, a practical advantage in real-world systems where only limited data may be observable at arrival.

5.4.1 Graphical Interpretation: Four-Feature Models

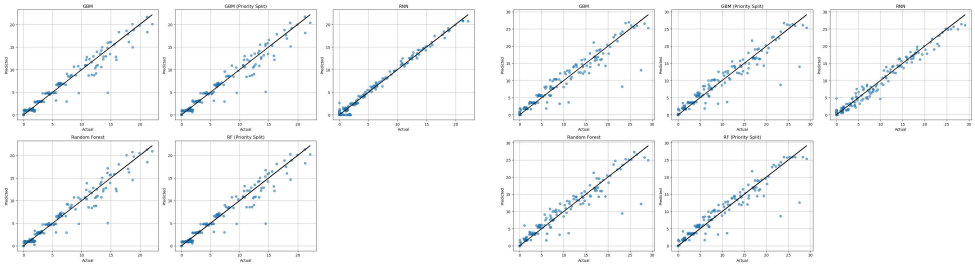
Figure 1 illustrates the predicted versus actual waiting times for a representative day (180 customers) under all three service disciplines when all four input features (priority, queue length, arrival time, and inter-arrival time) are used.

5.4.2 Graphical Interpretation: Two-Feature Models

Figure 2 presents the predicted versus actual waiting times using only two input features-priority and queue length. Despite the reduced feature set, the prediction curves remain smooth and well-aligned with actual waiting-time trajectories.

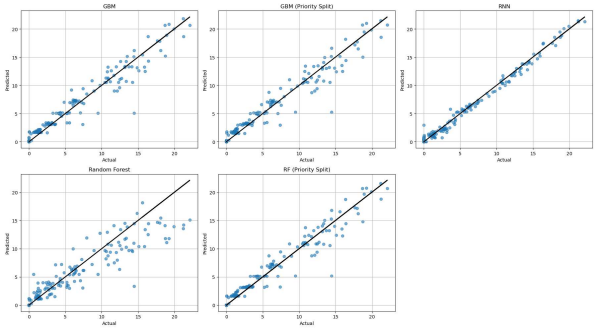
5.5 Overall Interpretation

Across all six experimental configurations (three service disciplines evaluated under both four-feature and two-feature models), the RNN consistently delivers the strongest



(a) Non-preemptive

(b) Preemptive-repeat



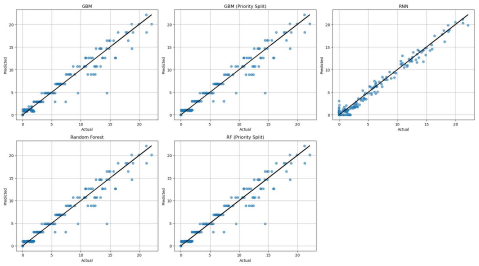
(c) Preemptive-resume

Fig. 1: Actual vs. predicted waiting times using four input features across the three service disciplines. Enlarged visualization for clarity.

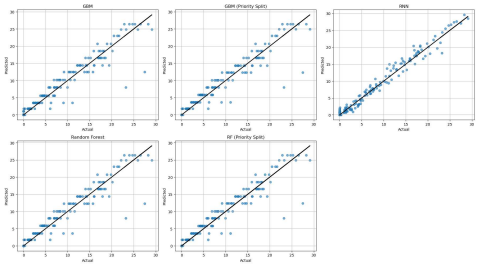
predictive performance. GBM, RF, and their priority-split variants also demonstrate stable and reliable behavior, with only modest degradation when the feature set is reduced.

A key insight emerging from these experiments is that: **Queue length and priority alone retain most of the predictive information required for waiting-time prediction.**

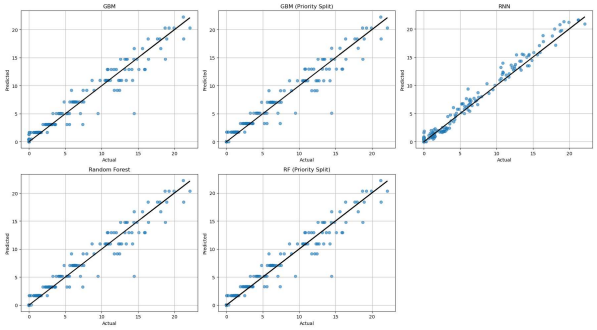
This finding substantially enhances the practical relevance of the proposed framework. In many operational environments-such as hospitals with pre-registration systems, call centers with digital ticketing, airport service counters, automotive service facilities, and online appointment platforms-the only readily observable attributes at arrival are the current queue length and the customer's priority class. The fact that



(a) Non-preemptive



(b) Preemptive-repeat



(c) Preemptive-resume

Fig. 2: Actual vs. predicted waiting times using only queue length and priority. Enlarged visualization highlights how well the models perform even under reduced feature input.

feasible predictions remain achievable under such minimal information requirements makes the approach highly deployable in real-time settings.

Overall, the unified evidence from all six tables and the graphical analyses shows that machine-learning models-particularly sequence-based models such as RNNs-provide a powerful and implementable solution for real-time waiting-time prediction in multi-priority, time-varying service systems.

6 Conclusion

This study focused on predicting waiting times in dynamic, multi-class priority queues under three service types: non-preemptive, preemptive-repeat, and preemptive-resume. Five machine learning models were tested: GBM, RF, their priority-split versions, and an RNN.

A secondary objective of this study was to examine whether explicitly embedding queuing-domain structure through priority-based splitting improves predictive performance. The results indicate that such modifications provide little benefit, suggesting that modern tree ensembles already capture priority effects effectively when priority is included as an input feature.

Among all evaluated models, the RNN performed the best. It handled time-dependent behavior better, especially in preemptive systems where interruptions and restarts occur. Tree-based models were stable and efficient, but less capable of capturing changes over time.

An important finding is that strong prediction accuracy can be achieved using only two inputs: **queue length at arrival** and **customer priority**. This is surprising because, in priority queues, waiting time (especially for low-priority customers) depends on future high-priority arrivals. Traditional methods require detailed system information to model this. In contrast, the learning models, especially the RNN, are able to capture these effects without being given any future information.

This is very useful in practice. Most real systems can easily record queue length and priority, but do not know exact arrival patterns or service distributions. Our results show that even with this limited information, accurate real-time prediction is possible.

Overall, RNNs are the most effective for this problem, while tree-based models remain reliable but less flexible. Also, explicitly forcing priority splits does not add

much value. These results support the use of simple, data-driven approaches for real-world priority-based service systems.

A Additional Analysis for the Preemptive-Repeat System

This appendix provides supplementary experiments for the machine learning models under the preemptive-repeat priority queue. The objective is to investigate model sensitivity, bias-variance behavior, and hyperparameter robustness beyond the predictive results reported in the main text.

A.1 Gradient Boosting Machine

A.1.1 Effect of the Number of Estimators

The predictive performance of Gradient Boosting Machines depends on the number of boosting stages used in the ensemble. To evaluate this sensitivity, the number of estimators was varied while keeping all other hyperparameters fixed.

Table 7: Effect of the number of estimators on GBM performance under the preemptive-repeat system.

Number of Estimators	Test R^2
50	0.9348
80	0.9357
100	0.9357
120	0.9359
150	0.9359
180	0.9359
200	0.9359
220	0.9359
250	0.9360

Figure 3 plots the corresponding relationship.

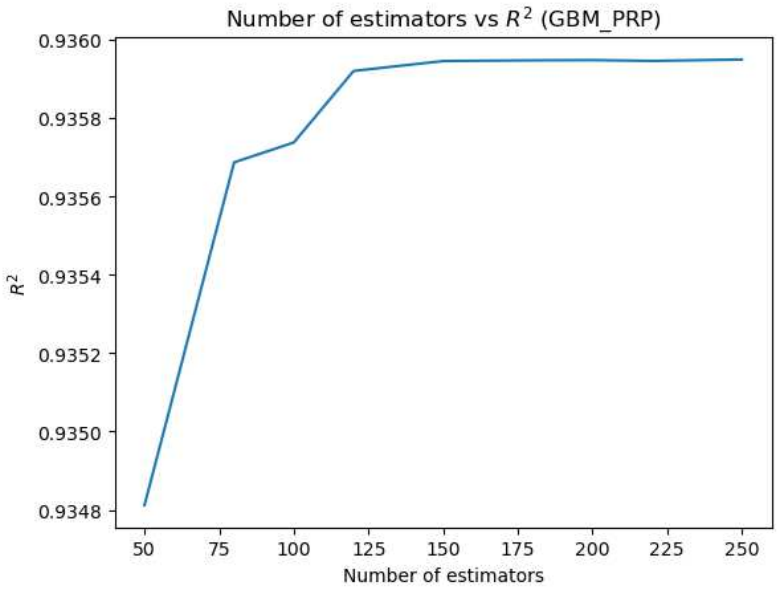


Fig. 3: Preemptive repeat - GBM - Number of estimators

The results show a gradual increase in predictive accuracy as the number of estimators increases. However, the improvement becomes negligible beyond approximately 120 estimators. The difference between 120 and 250 estimators is less than 0.001 in terms of R^2, indicating that the ensemble has effectively converged. Consequently, increasing the ensemble size beyond this point yields little practical benefit while increasing computational cost.

A.1.2 Bias-Variance Analysis

To examine the bias-variance trade-off, the maximum depth of the regression trees was varied while all other hyperparameters remained unchanged. Shallow trees generally exhibit higher bias and lower variance, whereas deeper trees reduce bias but may increase variance and overfitting.

Figure 4 presents the corresponding training and testing curves.

Table 8: Training and testing R^2 values for different tree depths under the preemptive-repeat system.

Maximum Depth	Training R^2	Testing R^2
2	0.9369	0.9315
3	0.9406	0.9357
5	0.9409	0.9359
8	0.9409	0.9359
10	0.9409	0.9359
12	0.9409	0.9359
15	0.9409	0.9359
18	0.9409	0.9359
20	0.9409	0.9359
22	0.9409	0.9359
25	0.9409	0.9359

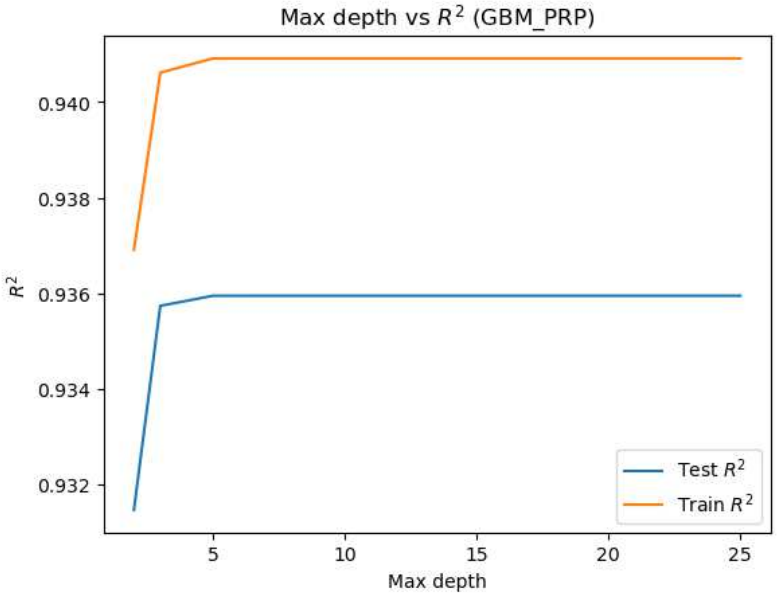


Fig. 4: Preemptive repeat - GBM - Max Depth

The results reveal that both training and testing performance improve substantially when the maximum depth increases from 2 to 5. Beyond depth 5, both curves

become nearly constant and remain very close to one another. The absence of a widening gap between training and testing performance indicates that the model does not suffer from significant overfitting.

These observations suggest that the waiting-time prediction problem exhibits a relatively simple nonlinear structure that can be captured effectively using moderate tree depths. Increasing tree complexity beyond this point provides no meaningful improvement in predictive performance. The close alignment of the training and testing curves further indicates a favorable balance between bias and variance.

A.2 Random Forest

A.2.1 Effect of the Number of Trees

Random Forest performance depends on the number of trees used in the ensemble. Increasing the number of trees generally reduces prediction variance by averaging across a larger collection of decorrelated decision trees. To investigate this effect, the number of trees was varied while keeping all remaining hyperparameters fixed.

Table 9: Effect of the number of trees on Random Forest performance under the preemptive-repeat system.

Number of Trees	Test R^2
50	0.9360
80	0.9359
100	0.9360
120	0.9359
150	0.9360
180	0.9359
200	0.9359
220	0.9359
250	0.9359

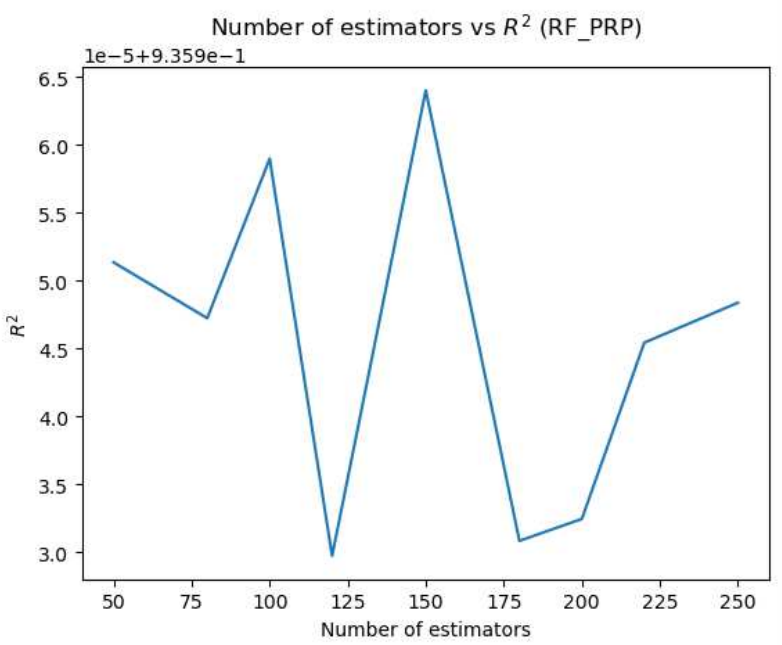


Fig. 5: Preemptive repeat - RF - Number of estimators

Figure 5 shows the corresponding relationship between ensemble size and predictive accuracy.

The results indicate that Random Forest performance is remarkably stable across all ensemble sizes considered. The variation in R^2 is extremely small, remaining close to 0.936 throughout the entire range from 50 to 250 trees. This suggests that the ensemble reaches convergence rapidly and that increasing the number of trees beyond approximately 50 provides little additional predictive benefit. Such behavior is consistent with the variance-reduction mechanism of bagging, where a sufficiently large ensemble already captures the dominant structure present in the data.

A.2.2 Bias-Variance Analysis

To study the bias-variance trade-off, the maximum tree depth was varied while maintaining all other parameters fixed. Shallow trees are expected to exhibit higher bias

due to limited model complexity, whereas deeper trees can reduce bias at the expense of increased variance.

Table 10: Training and testing R^2 values for different tree depths under the preemptive-repeat system.

Maximum Depth	Training R^2	Testing R^2
2	0.7583	0.7658
3	0.8960	0.8892
5	0.9404	0.9347
8	0.9409	0.9359
10	0.9409	0.9359
12	0.9409	0.9359
15	0.9409	0.9359
18	0.9409	0.9360
20	0.9409	0.9360
22	0.9409	0.9359
25	0.9409	0.9360

Figure 6 presents the corresponding training and testing curves.

The effect of tree depth is substantially more pronounced than the effect of ensemble size. At depths 2 and 3, both training and testing R^2 values are considerably lower, indicating underfitting due to excessive model bias. Increasing the depth to 5 results in a dramatic improvement in predictive accuracy, with testing R^2 increasing from approximately 0.89 to 0.935.

Beyond depth 8, both the training and testing curves stabilize and remain nearly constant. Importantly, the difference between training and testing performance remains very small across all larger depths. This indicates that the Random Forest model does not suffer from substantial overfitting even when deeper trees are used.

The observed behavior is characteristic of Random Forest ensembles. While individual deep trees may have high variance, averaging across many bootstrap samples effectively controls variance, allowing the model to benefit from reduced bias without a corresponding increase in overfitting. Consequently, the Random Forest achieves a favorable bias-variance balance once moderate tree depths are reached.

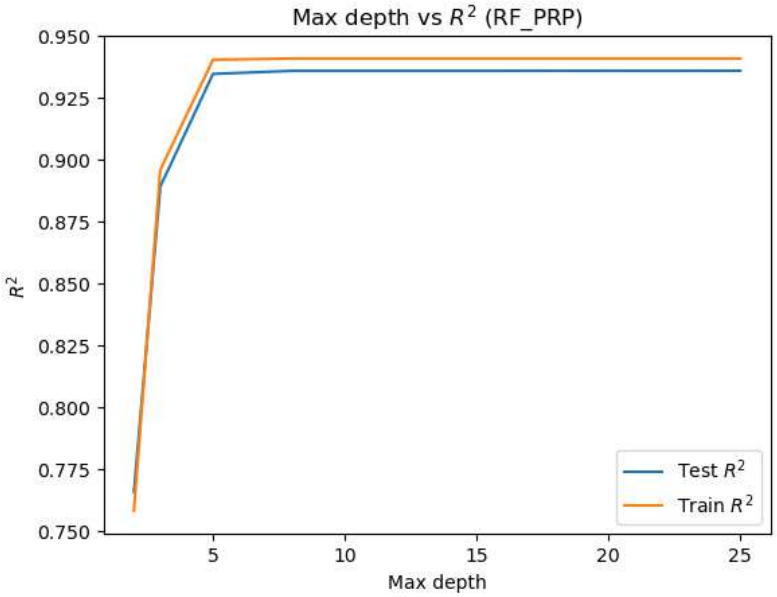


Fig. 6: Preemptive repeat - RF - Max Depth

A.3 Recurrent Neural Network

A.3.1 Sensitivity to the Number of LSTM Units

The predictive performance of recurrent neural networks depends strongly on the capacity of the hidden state representation. To investigate the effect of model complexity, the number of units in the first LSTM layer was varied while keeping all other architectural and training parameters fixed.

Table 11: Effect of the number of LSTM units on RNN performance under the preemptive-repeat system.

LSTM Units	R^2	MAE	RMSE
32	0.9781	1.1233	1.6493
64	0.9868	0.8719	1.2792
128	0.9778	1.1900	1.6626
256	0.9796	1.0576	1.5943

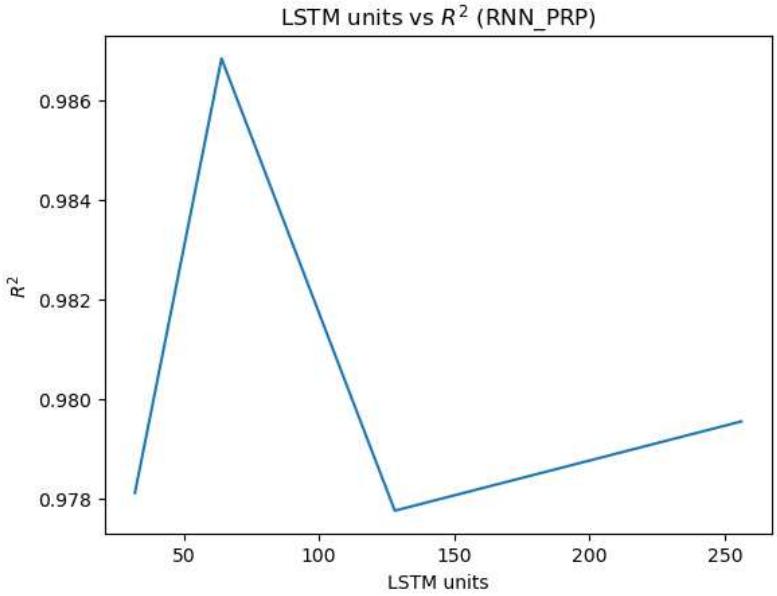


Fig. 7: Preemptive repeat - RNN

Figure 7 illustrates the relationship between the number of LSTM units and predictive performance.

The results demonstrate that model capacity has a significant influence on prediction accuracy. Increasing the number of LSTM units from 32 to 64 leads to a substantial improvement in all evaluation metrics. The coefficient of determination increases from 0.9781 to 0.9868, while both MAE and RMSE decrease considerably.

Interestingly, further increases in model size do not improve performance. Both the 128-unit and 256-unit architectures perform worse than the 64-unit configuration. This suggests that excessively large recurrent representations introduce unnecessary model complexity without providing additional useful information for the waiting-time prediction task.

The observed behavior is consistent with the classical trade-off between model capacity and generalization. While a small network may not possess sufficient representational power to capture the temporal dependencies generated by customer arrivals

and service interruptions, an excessively large network may learn noise present in the training data and therefore generalize less effectively.

A.3.2 Selection of the Final Architecture

Among all tested configurations, the model containing 64 LSTM units achieved the highest predictive accuracy.

$$R^2 = 0.9868,$$

with corresponding

$$\text{MAE} = 0.8719, \qquad \text{RMSE} = 1.2792.$$

Consequently, the 64-unit architecture was selected as the final RNN model used throughout the main experiments.

The results indicate that the waiting-time prediction problem possesses a moderate level of temporal complexity. A hidden representation consisting of 64 memory units appears sufficient to capture the essential interactions among customer priority, queue congestion, arrival dynamics, and service interruptions. Increasing the network size beyond this level does not yield additional predictive benefit and may slightly reduce generalization performance.

A.3.3 Comparison with Tree-Based Models

The hyperparameter analysis provides additional insight into the superior performance of the RNN observed in the main results. Unlike GBM and Random Forest, which operate on static feature vectors, the RNN explicitly models the temporal ordering of customer arrivals through its recurrent memory structure.

The optimal 64-unit architecture achieves a substantially higher R^2 than both ensemble-tree approaches, indicating that temporal information plays an important

role in predicting waiting times under preemptive service disciplines. This advantage becomes particularly important when interruptions, restarts, and future congestion effects influence customer waiting times.

Overall, the sensitivity analysis confirms that the RNN is capable of exploiting sequential dependencies that are difficult to represent through static tree-based models, thereby explaining its consistently superior predictive performance throughout the study.

B Additional Analysis for the Preemptive-Resume System

B.1 Gradient Boosting Machine

B.1.1 Effect of the Number of Estimators

To evaluate the sensitivity of the Gradient Boosting Machine (GBM) to ensemble size, the number of boosting stages was varied while all remaining hyperparameters were held fixed. The resulting test-set R^2 values are reported in Table 12.

Table 12: Effect of the number of estimators on GBM performance under the preemptive-resume system.

Number of Estimators	Test R^2
50	0.9249
80	0.9267
100	0.9270
120	0.9270
150	0.9272
180	0.9273
200	0.9273
220	0.9273
250	0.9273

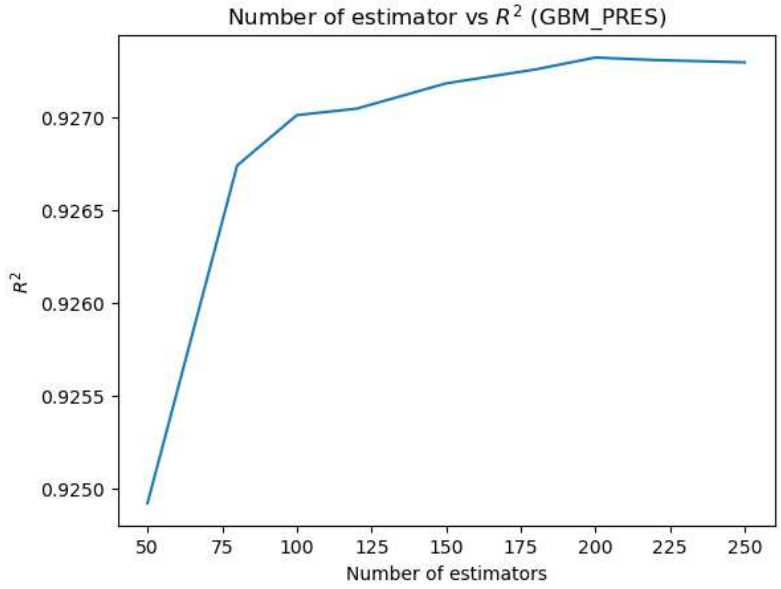


Fig. 8: Preemptive resume - GBM - Number of estimators

Figure 8 illustrates the corresponding relationship between ensemble size and predictive performance.

The results show a gradual improvement in predictive accuracy as additional boosting stages are introduced. The largest gains occur between 50 and 100 estimators, after which the performance curve begins to flatten. Beyond approximately 150 estimators, the improvements become negligible, indicating that the boosting process has effectively converged.

This behavior suggests that a relatively moderate ensemble size is sufficient to capture the dominant nonlinear relationships governing waiting times in the preemptive-resume system. Additional boosting stages increase computational cost while providing only marginal predictive benefit.

B.1.2 Bias-Variance Analysis

To investigate the bias-variance trade-off, the maximum depth of the regression trees was varied while keeping all remaining hyperparameters fixed. Table 13 reports the corresponding training and testing R^2 values.

Table 13: Training and testing R^2 values for different tree depths under the preemptive-resume system.

Maximum Depth	Training R^2	Testing R^2
2	0.9325	0.9218
3	0.9376	0.9270
5	0.9411	0.9270
8	0.9550	0.9240
10	0.9691	0.9211
12	0.9819	0.9160
15	0.9937	0.9050
18	0.9977	0.8943
20	0.9982	0.8879
22	0.9983	0.8815
25	0.9983	0.8750

Figure 9 presents the corresponding training and testing performance curves.

A pronounced bias-variance trade-off is observed. For shallow trees (depths 2-5), both training and testing performance improve as model complexity increases, indicating a reduction in bias. The highest testing performance is achieved at depths between 3 and 5, where the model attains an R^2 value of approximately 0.927.

Beyond depth 5, a substantial divergence between the training and testing curves emerges. The training R^2 continues to increase rapidly, eventually approaching 1.0, while the testing R^2 decreases steadily. At depth 25, the training R^2 exceeds 0.998, whereas the testing R^2 falls below 0.875.

This behavior is characteristic of overfitting. As tree complexity increases, the GBM increasingly memorizes idiosyncratic patterns and noise present in the training data rather than learning the underlying queuing dynamics. Consequently, predictive performance on unseen observations deteriorates.

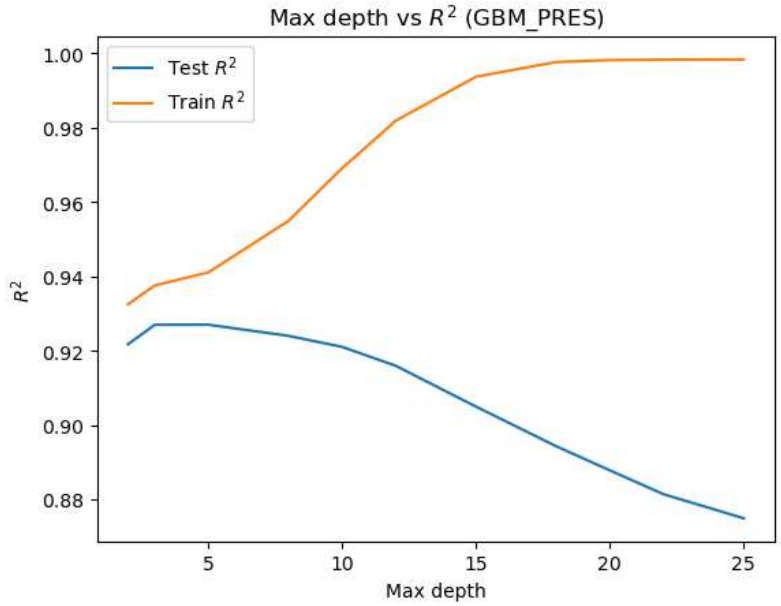


Fig. 9: Preemptive resume - GBM - Max depth

The overfitting effect is considerably stronger than that observed for the Random Forest model. This difference arises because boosting sequentially fits residual errors and therefore has a greater tendency to learn noise when individual trees become excessively deep. In contrast, Random Forest mitigates variance through averaging across multiple independently constructed trees.

B.2 Random Forest

B.2.1 Effect of the Number of Trees

To investigate the sensitivity of the Random Forest model to ensemble size, the number of trees was varied while all remaining hyperparameters were kept fixed. The resulting test-set R^2 values are reported in Table 14.

Figure 10 illustrates the corresponding relationship between the number of trees and predictive performance.

Table 14: Effect of the number of trees on Random Forest performance under the preemptive-resume system.

Number of Trees	Test R^2
100	0.9248
150	0.9248
180	0.9248
200	0.9247
220	0.9249
250	0.9248
300	0.9250
350	0.9248
400	0.9248

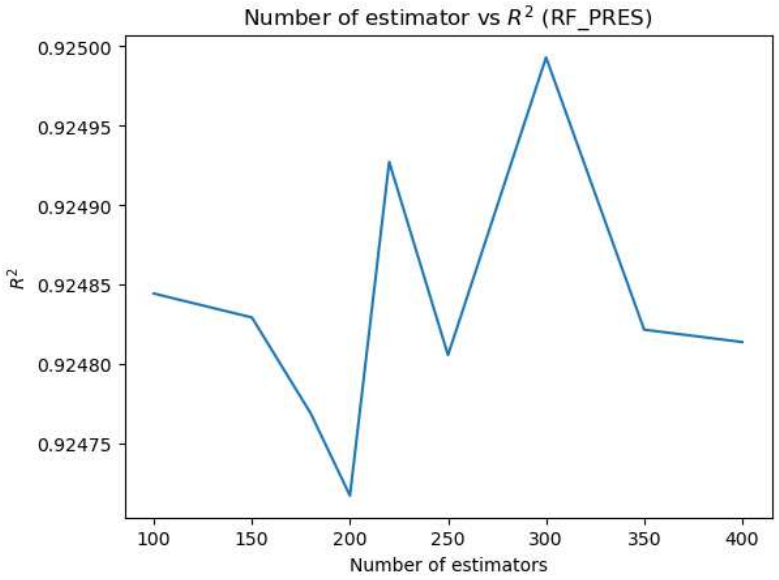


Fig. 10: Preemptive resume - RF - Number of estimators

The results indicate that the Random Forest model is largely insensitive to the exact number of trees once a sufficiently large ensemble has been constructed. Across all tested values, the coefficient of determination remains close to 0.925, with only negligible variation. The best performance is obtained with 300 trees, although the

improvement over smaller ensembles is extremely small. This behavior is consistent with the theoretical properties of bagging-based methods, where prediction variance decreases rapidly as additional trees are added and eventually reaches a stable plateau.

B.2.2 Bias-Variance Analysis

To examine the bias-variance trade-off, the maximum tree depth was varied while keeping all remaining hyperparameters fixed. Table 15 reports the resulting training and testing R^2 values.

Table 15: Training and testing R^2 values for different tree depths under the preemptive-resume system.

Maximum Depth	Training R^2	Testing R^2
2	0.7391	0.7320
3	0.8870	0.8697
5	0.9351	0.9235
8	0.9404	0.9270
10	0.9463	0.9263
12	0.9545	0.9250
15	0.9675	0.9221
18	0.9775	0.9189
20	0.9820	0.9170
22	0.9847	0.9157
25	0.9870	0.9140

Figure 11 presents the corresponding training and testing performance curves.

Unlike the preemptive-repeat system, a clearer bias-variance trade-off is observed under the preemptive-resume discipline. When the maximum depth increases from 2 to 8, both training and testing performance improve substantially, indicating a reduction in model bias. The highest testing performance is achieved at depth 8, where the model attains an R^2 value of approximately 0.927.

Beyond depth 8, the training R^2 continues to increase steadily, eventually reaching values close to 0.99. In contrast, the testing R^2 gradually decreases as depth increases. This divergence between the training and testing curves is a classical indication of

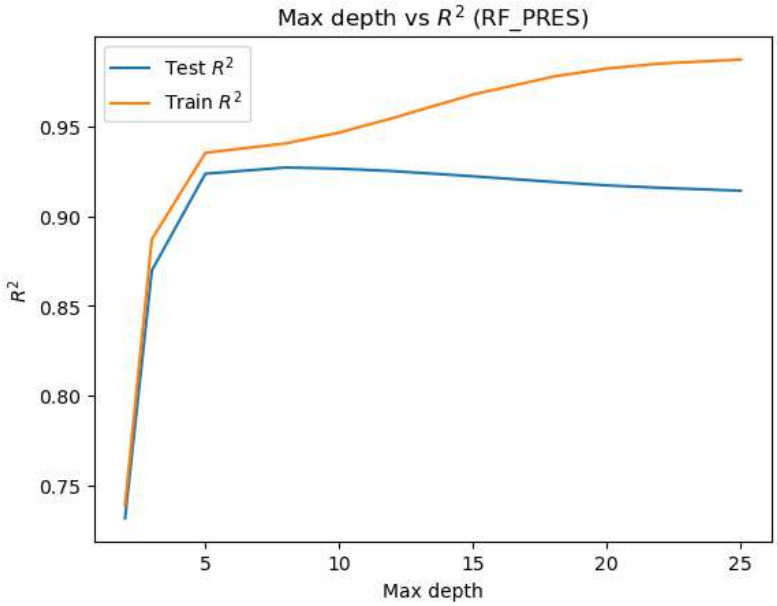


Fig. 11: Preemptive resume - RF - Max depth

overfitting. As the trees become increasingly deep, the model begins to fit noise and random fluctuations present in the training data rather than the underlying waiting-time structure.

The results therefore suggest that moderate tree depths provide the most effective balance between bias and variance for the preemptive-resume waiting-time prediction problem. In particular, depths between 5 and 10 appear to achieve the strongest generalization performance.

B.3 Recurrent Neural Network

B.3.1 Sensitivity to the Number of LSTM Units

To evaluate the influence of network capacity on predictive performance, the number of units in the first LSTM layer was varied while keeping all remaining architectural and training parameters unchanged. The resulting performance metrics are summarized in Table 16.

Table 16: Effect of the number of LSTM units on RNN performance under the preemptive-resume system.

LSTM Units	R^2	MAE	RMSE
32	0.9812	0.7313	1.0491
64	0.9808	0.7731	1.0612
128	0.9872	0.5631	0.8646
256	0.9783	0.8245	1.1277

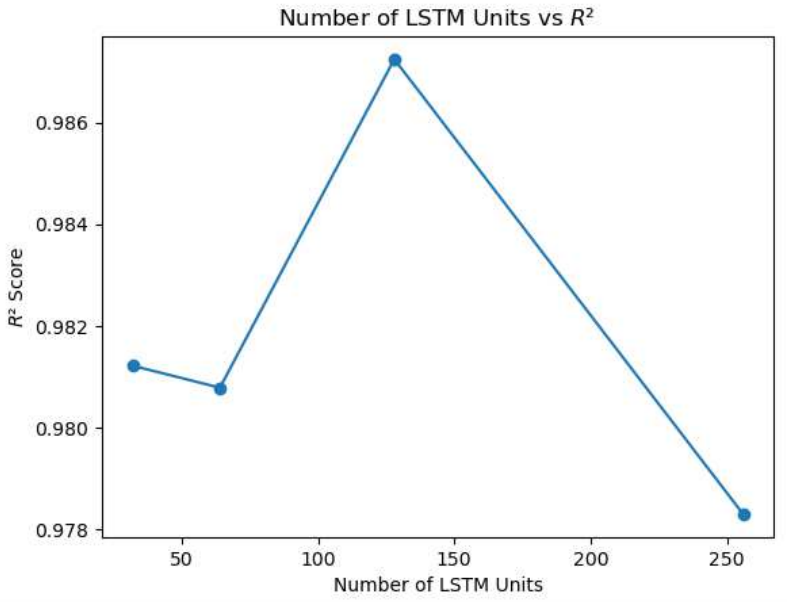


Fig. 12: Preemptive resume - RNN

Figure 12 illustrates the relationship between the number of LSTM units and predictive performance.

The results indicate that network capacity has a substantial influence on prediction accuracy. Increasing the number of LSTM units from 32 to 128 leads to a noticeable improvement in all evaluation metrics. The coefficient of determination increases from 0.9812 to 0.9872, while both MAE and RMSE decrease significantly.

However, further increasing the network size to 256 units causes performance to deteriorate. Although larger networks possess greater representational capacity, they

also introduce additional trainable parameters and become more susceptible to overfitting. The decline in testing performance suggests that the additional complexity is not justified by the amount of information contained in the waiting-time data.

These observations indicate the existence of an optimal model capacity for the preemptive-resume prediction problem. Networks that are too small fail to capture the full temporal dynamics of the queuing process, whereas excessively large networks tend to learn noise rather than meaningful system behavior.

B.3.2 Selection of the Final Architecture

Among all tested configurations, the model containing 128 LSTM units achieved the strongest predictive performance:

$$R^2 = 0.9872,$$

with

$$\text{MAE} = 0.5631, \qquad \text{RMSE} = 0.8646.$$

Consequently, the 128-unit architecture was selected as the final RNN configuration used for the preemptive-resume experiments.

Interestingly, the optimal architecture for the preemptive-resume system differs from that observed in the preemptive-repeat setting. While the preemptive-repeat system achieved its best performance using 64 LSTM units, the preemptive-resume system benefits from a larger hidden representation. This difference is likely due to the additional complexity introduced by residual service times. In the preemptive-resume discipline, interrupted customers retain partially completed service, causing future waiting times to depend not only on queue lengths and priorities but also on the remaining service requirements of customers currently in the system.

As a result, the temporal dependencies governing waiting times are more intricate than those observed under preemptive-repeat service. The larger hidden state appears to provide sufficient representational capacity to capture these additional dependencies.

C Additional Analysis for the Non-Preemptive System

C.1 Gradient Boosting Machine

C.1.1 Effect of the Number of Estimators

To investigate the influence of ensemble size on predictive performance, the number of boosting stages was varied while all remaining hyperparameters were kept fixed. The resulting test-set R^2 values are reported in Table 17.

Table 17: Effect of the number of estimators on GBM performance under the non-preemptive system.

Number of Estimators	Test R^2
50	0.940
80	0.942
100	0.942
120	0.942
150	0.942
160	0.942
180	0.942
200	0.942
220	0.942
250	0.942

Figure 13 illustrates the corresponding relationship between the number of estimators and predictive performance.

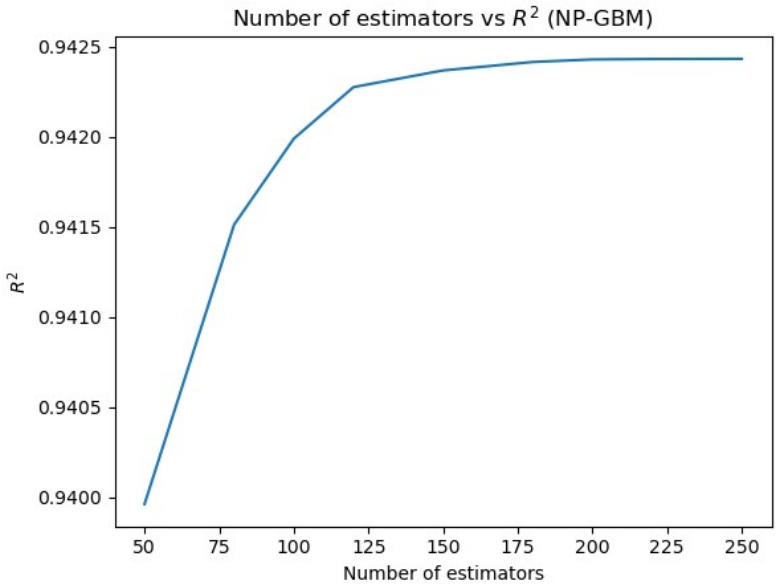


Fig. 13: Non preemptive - GBM - Number of estimators

The results indicate that predictive accuracy improves rapidly when the number of estimators increases from 50 to approximately 100. Beyond this point, the performance curve becomes essentially flat, with the coefficient of determination remaining close to $R^2 = 0.942$ for all larger ensembles.

This behavior suggests that the boosting procedure converges quickly and that the dominant nonlinear relationships governing waiting times are captured using a relatively small number of boosting stages. Additional estimators increase computational cost while providing negligible improvement in predictive accuracy.

C.1.2 Bias-Variance Analysis

To examine the bias-variance trade-off, the maximum depth of the regression trees was varied while keeping all remaining parameters fixed. The corresponding training and testing R^2 values are summarized in Table 18.

Figure 14 presents the corresponding training and testing performance curves.

Table 18: Training and testing R^2 values for different tree depths under the non-preemptive system.

Maximum Depth	Training R^2	Testing R^2
5	0.950	0.942
8	0.950	0.942
10	0.950	0.942
12	0.950	0.942
15	0.950	0.942
16	0.950	0.942
18	0.950	0.942
20	0.950	0.942
22	0.950	0.942
25	0.950	0.942

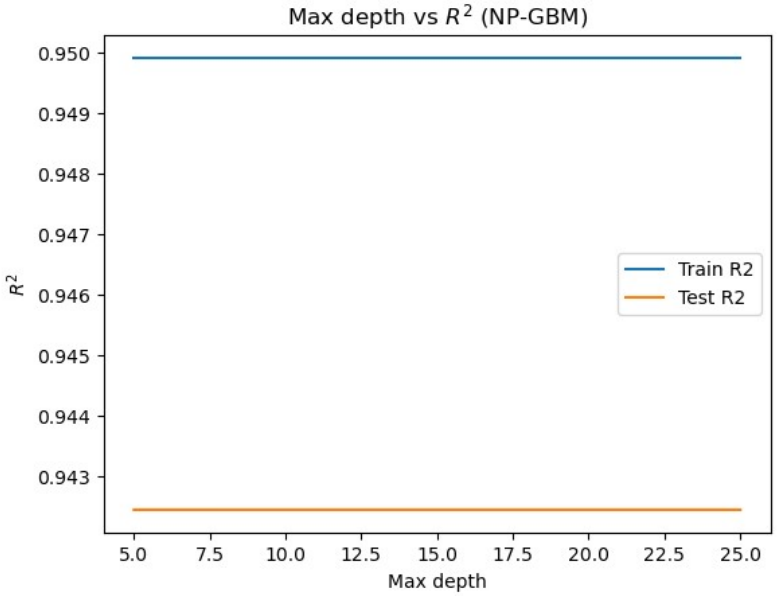


Fig. 14: Non preemptive - GBM - Max depth

The training and testing curves remain nearly constant across the entire range of tree depths considered. Training performance stabilizes at approximately $R^2 = 0.950$, while testing performance remains close to $R^2 = 0.942$. The gap between the two curves is small and does not increase as model complexity grows.

Unlike the preemptive-resume system, where deeper trees produced noticeable overfitting, the non-preemptive system exhibits highly stable behavior. Increasing tree depth neither improves nor degrades predictive accuracy in a meaningful way. This indicates that the underlying waiting-time relationships are comparatively smooth and can be captured effectively using moderate tree complexity.

The absence of a widening gap between training and testing performance suggests that the GBM achieves a favorable balance between bias and variance across a broad range of depths.

C.2 Random Forest

C.2.1 Effect of the Number of Trees

To evaluate the sensitivity of the Random Forest model to ensemble size, the number of trees was varied while keeping all other hyperparameters fixed. The resulting test-set R^2 values are reported in Table 19.

Table 19: Effect of the number of trees on Random Forest performance under the non-preemptive system.

Number of Trees	Test R^2
100	0.94230
150	0.94233
180	0.94231
200	0.94232
220	0.94234
250	0.94232
300	0.94232
350	0.94232
400	0.94232

Figure 15 illustrates the relationship between ensemble size and predictive performance.

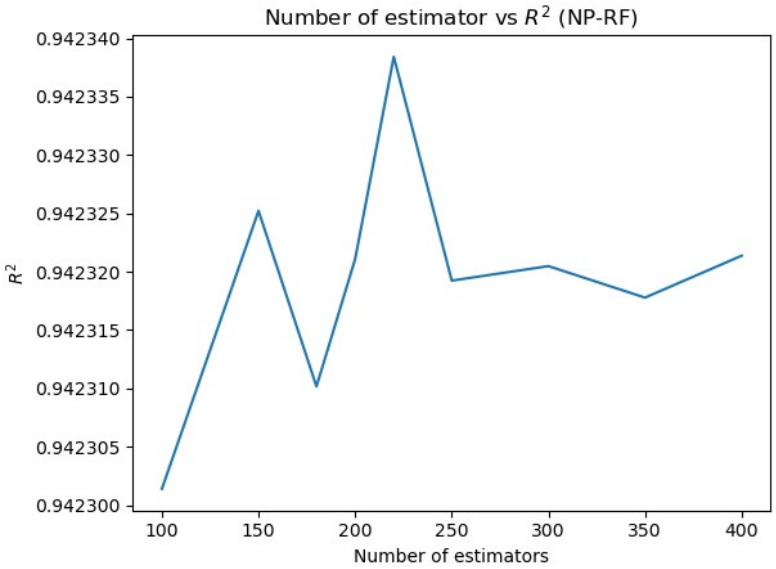


Fig. 15: Non preemptive - RF - Number of estimators

The results indicate that Random Forest performance is essentially insensitive to the number of trees once a moderate ensemble size has been reached. Across all tested configurations, the coefficient of determination remains extremely stable at approximately $R^2 = 0.9423$. The difference between the best and worst performing configurations is less than 0.0001, indicating that the ensemble converges rapidly and that additional trees provide negligible improvement.

This stability suggests that the dominant waiting-time patterns in the non-preemptive system are relatively easy for the ensemble to learn. Consequently, even moderately sized forests are sufficient to achieve near-optimal predictive performance.

C.2.2 Bias-Variance Analysis

To investigate the bias-variance trade-off, the maximum tree depth was varied while all remaining hyperparameters were held fixed. The corresponding training and testing R^2 values are presented in Table 20.

Figure 16 presents the corresponding training and testing performance curves.

Table 20: Training and testing R^2 values for different tree depths under the non-preemptive system.

Maximum Depth	Training R^2	Testing R^2
2	0.7482	0.7436
3	0.8905	0.8691
5	0.9491	0.9384
8	0.9499	0.9422
10	0.9499	0.9423
12	0.9499	0.9423
15	0.9499	0.9423
18	0.9499	0.9423
20	0.9499	0.9423
22	0.9499	0.9423
25	0.9499	0.9423

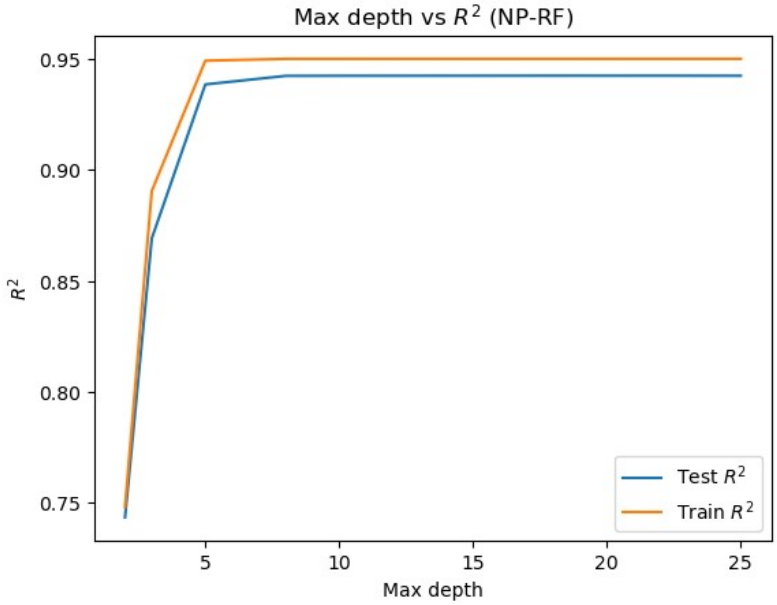


Fig. 16: Non preemptive - RF - Max depth

A clear reduction in bias is observed as tree depth increases from 2 to 8. Both training and testing performance improve substantially over this range, indicating that deeper trees are better able to capture the nonlinear relationships governing customer waiting times.

Unlike the preemptive-resume system, however, no significant overfitting is observed. Beyond depth 8, both the training and testing curves become nearly flat and remain closely aligned. The training R^2 stabilizes near 0.950, while the testing R^2 stabilizes near 0.942. The gap between the two curves remains small and approximately constant across all larger depths.

This behavior indicates that the Random Forest model achieves an effective balance between bias and variance once moderate tree depths are reached. Additional complexity neither improves nor degrades predictive performance in a meaningful way.

C.3 Recurrent Neural Network

C.3.1 Sensitivity to the Number of LSTM Units

To investigate the influence of network capacity on predictive performance, the number of units in the first LSTM layer was varied while all remaining architectural and training parameters were held fixed. The resulting performance metrics are summarized in Table 21.

Table 21: Effect of the number of LSTM units on RNN performance under the non-preemptive system.

LSTM Units	R^2	MAE	RMSE
32	0.9765	0.8386	1.1428
64	0.9844	0.6884	0.9302
128	0.9711	0.9541	1.2675
256	0.9775	0.8449	1.1192

Figure 17 illustrates the relationship between the number of LSTM units and predictive performance.

The results indicate that network capacity has a substantial influence on prediction accuracy. Increasing the number of LSTM units from 32 to 64 produces a noticeable

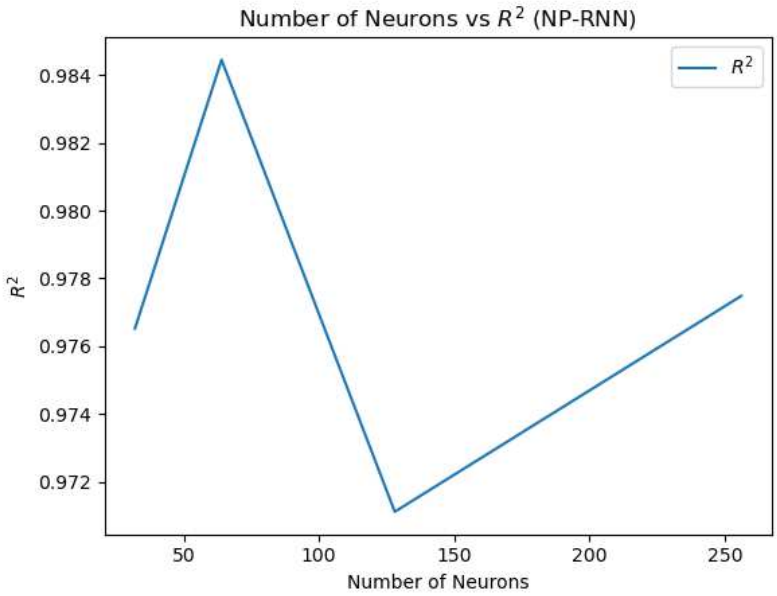


Fig. 17: Non preemptive - RNN

improvement in all evaluation metrics. The coefficient of determination increases from 0.9765 to 0.9844, while both MAE and RMSE decrease significantly.

However, further increases in network size do not improve performance. Both the 128-unit and 256-unit architectures perform worse than the 64-unit configuration. In particular, the 128-unit model exhibits the lowest predictive accuracy among all tested configurations. These results suggest that excessively large recurrent representations introduce unnecessary complexity without providing additional useful information for the waiting-time prediction task.

This behavior is consistent with the classical trade-off between model complexity and generalization. While a sufficiently large hidden representation is required to capture temporal dependencies, overly complex architectures may learn noise present in the training data and consequently generalize less effectively to unseen observations.

C.3.2 Selection of the Final Architecture

Among all tested configurations, the model containing 64 LSTM units achieved the highest predictive performance:

$$R^2 = 0.9844,$$

with

$$\text{MAE} = 0.6884, \qquad \text{RMSE} = 0.9302.$$

Consequently, the 64-unit architecture was selected as the final RNN configuration used for the non-preemptive experiments.

Interestingly, the optimal architecture for the non-preemptive system is identical to that observed in the preemptive-repeat setting, where 64 LSTM units also provided the best performance. This suggests that the temporal dependencies present in these two service disciplines possess comparable complexity. Since interrupted customers either do not exist (non-preemptive) or restart service from the beginning (preemptive-repeat), the system dynamics remain relatively straightforward compared with the preemptive-resume discipline.

As a result, a moderate-sized hidden representation appears sufficient to capture the essential interactions among arrivals, queue congestion, customer priority, and waiting times.

Declaration of generative AI and AI-assisted technologies in the manuscript preparation process

During the preparation of this manuscript, the authors used ChatGPT (OpenAI, 2025) to improve readability and linguistic clarity. The authors reviewed, edited, and

verified all content to ensure accuracy and integrity. All conceptual developments, analysis, and conclusions are the authors' own work.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data Availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Author Contributions

T.G. Deepak and Sahil Singh Bhandari contributed equally to this work. Both authors were involved in the conception and design of the study, development of the methodology, data generation and analysis, interpretation of results, and manuscript preparation. Both authors read and approved the final manuscript.

References

Aksin Z, Armony M, Mehrotra V (2007) The modern call center: A multi-disciplinary perspective on operations management research. Production and Operations Management 16(6):665–688. https://doi.org/10.1111/j.1937-5956.2007.tb00219.x

Baron O, Krass D, Senderovich A, et al (2024) Supervised machine learning for solving the gi/gi/1 queue. INFORMS Journal on Computing 36(3):766–786. https://doi.org/10.1287/ijoc.2022.0263

Bhandari SS, Deepak TG (2026) Simulation-based transient prediction and cost optimization in multi-server queueing systems using a hybrid autoregressive neural model. Applied Soft Computing 201:115563. https://doi.org/10.1016/j.asoc.2026.115563

Breiman L (2001) Random forests. Machine Learning 45(1):5–32

Brown L, Gans N, Mandelbaum A, et al (2005) Statistical analysis of a telephone call center: A queueing-science perspective. Manufacturing & Service Operations Management 7(1):20–45

Cayirli T, Veral E (2003) Outpatient appointment scheduling in health care: A review. Health Care Management Science 6(1):1–28

Friedman JH (2001) Greedy function approximation: A gradient boosting machine. Annals of Statistics 29(5):1189–1232

Gans N, Koole G, Mandelbaum A (2003) Telephone call centers: Tutorial, review, and research prospects. Manufacturing & Service Operations Management 5(2):79–141. https://doi.org/10.1287/msom.5.2.79.16071

Goodfellow I, Bengio Y, Courville A (2016) Deep Learning. MIT Press, see Chapter 10: Sequence Modeling with Recurrent Networks

Gross D, Shortle JF, Thompson JM, et al (2008) Fundamentals of Queueing Theory, 4th edn. Wiley

Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Computation 9(8):1735–1780

Ibrahim R (2018) Sharing delay information in service systems: A literature survey. Queueing Systems 89(1–2):49–79. https://doi.org/10.1007/s11134-018-9577-y

Ibrahim R, Armony M, Bassamboo A (2017) Does the past predict the future? the case of delay announcements in service systems. Management Science 63(6):1762–1780. https://doi.org/10.1287/mnsc.2015.2412

McGuire F, Noronha M, Sammon D (2022) Predicting patient waiting times in healthcare: A systematic review of data-driven approaches. Artificial Intelligence in Medicine 129:102315. https://doi.org/10.1016/j.artmed.2022.102315

Parekh AK, Gallager RG (1993) A generalized processor sharing approach to flow control in integrated services networks: the single-node case. Performance Evaluation 20(1-3):81–103

Sun Y, Heng BH, Tay SY, et al (2020) Predicting hospital emergency department wait times using machine learning models. International Journal of Medical Informatics 134:104042. https://doi.org/10.1016/j.ijmedinf.2019.104042

Takagi H (1991) Queueing Analysis, Volume 1: Vacation and Priority Systems. North-Holland, Amsterdam

Whitt W (1991) The pointwise stationary approximation for queues. Operations Research 39(3):464–483